Theoretical Computer Science 412 (2011) 1081–1091

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Bicriteria scheduling concerned with makespan and total completion time subject to machine availability constraints

Yumei Huo [a,*], Hairong Zhao [b]

[a] Department of Computer Science, College of Staten Island, CUNY, Staten Island, NY 10314, USA

[b] Department of Mathematics, Computer Science & Statistics, Purdue University, Hammond, IN 46323, USA

ARTICLE INFO

Article history:
Received 20 September 2010
Received in revised form 22 November 2010
Accepted 3 December 2010
Communicated by D.-Z. Du

Keywords:
Bicriteria scheduling
Machine availability constraint
Makespan
Total completion time
Polynomial time algorithms
Two machine

ABSTRACT

In the past, research on multiple criteria scheduling assumes that the number of available machines is fixed during the whole scheduling horizon and research on scheduling with limited machine availability assumes that there is only a single criterion that needs to be optimized. Both assumptions may not hold in real life. In this paper we simultaneously consider both bicriteria scheduling and scheduling with limited machine availability. We focus on two parallel machines' environment and the goal is to find preemptive schedules to optimize both makespan and total completion time subject to machine availability. Our main contribution in this paper is that we showed three bicriteria scheduling problems are in P by providing polynomial time optimal algorithms.

1. Introduction

Over the past couple of decades two very active areas in manufacturing and operations management research have been multicriteria scheduling and scheduling subject to machine availability constraints. While a lot of work has been done in both areas described above, research in these two areas has been conducted independently from one another.

In this paper, we simultaneously consider bicriteria scheduling and scheduling with limited machine availability. We focus on preemptive schedules on two parallel machines' environment. We are concerned with makespan and total completion time. The research in this paper is motivated not only by the lack of research results in this area, but also by its important applications in reality. The makespan and total completion time are two objectives of considerable interest. Minimizing makespan can ensure a good balance of the load among the machines and minimizing the total completion time can minimize the inventory holding costs. It is quite common that the manufacturers wish to minimize both objectives. On the other hand, each machine may have a maintenance period during which it cannot process any jobs. The production manager has to decide: with limited machine availability due to the preventive maintenance or periodical repair, how jobs should be scheduled in order to optimize both makespan and the total completion time?

Let us first introduce some notations. We use $J = \{J_1, J_2, \ldots, J_n\}$ to denote a set of n jobs. Each job J_i has a processing time p_i. Without loss of generality, the processing times of jobs are assumed to be an integer. A job can be preempted by another job or interrupted by machine unavailability and resumed later on any available machine. Let S be a feasible schedule of these jobs, the completion time of job J_i in schedule S is denoted by $C_i(S)$. If S is clear from the context, we will use C_i for short. The makespan of S is $C_{\max}(S) = \max\{C_i(S)\}$, and the total completion time of S is $\sum C_j(S)$. We will use $C^*_{\max}$ and C^*

* Corresponding author. Tel.: +1 7189822841; fax: +1 7189822856.
E-mail addresses: huo@mail.csi.cuny.edu (Y. Huo), hairong@calumet.purdue.edu (H. Zhao).

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.12.007

to denote the minimum makespan and minimum total completion time, respectively. The goal is to schedule the set of jobs on two parallel machines so as to minimize $\sum C_i$ subject to the condition that $C_{\max}$ is optimized, denoted by $\sum C_j/C_{\max}$; or to minimize $C_{\max}$ subject to the constraint that $\sum C_i$ is optimized, denoted by $C_{\max}/\sum C_j$.

To denote our problems, we extend the 3-field notation $\alpha \mid \beta \mid \gamma$ introduced by Graham et al. [4]. The first two problems we study assume that one machine is always available, and the problems are denoted by $P_{1,1} \mid r-a, prmt \mid \sum C_j/C_{\max}$ and $P_{1,1} \mid r-a, prmt \mid C_{\max}/\sum C_j$, respectively. We say an interval to be a *zero-availability interval* when both machines are not available. Since we are considering preemptive schedules, it is easy to see that for two machines, if there is no zero-availability interval, then the machine environment can be treated as if there is one machine always available. Note that it is practical to assume that there is no *zero-availability interval* since the preventive maintenance or periodical repair is usually done on a rotation basis instead of maintaining or repairing all the machines simultaneously. We also study a problem when there is a zero-availability interval; and the problem is denoted by $P_2 \mid r-a, prmt \mid C_{\max}/\sum C_j$.

Literature review. So far there is no research paper on multicriteria scheduling with limited machine availability constraints. In the following we will review the relevant results in the area of bicriteria scheduling and in the area of scheduling subject to limited machine availability. We will survey the results concerned with makespan and total completion only. For details about multicriteria scheduling, see [3,2,13,6]. For details about scheduling with limited machine availability, see [12,11].

Research on parallel machine multicriteria scheduling problems has not been adequately dealt with in the literature. Gupta et al. [5] proposes an exponential algorithm to solve optimally the bicriteria problem of minimizing the weighted sum of makespan and mean flowtime on two identical parallel machines. When preemption is allowed, $P \mid prmt \mid C_{\max}/\sum C_j$ and $P \mid prmt \mid \sum C_j/C_{\max}$ are polynomially solved by Leung and Young in [7], and Leung and Pinedo in [8], respectively.

With limited machine availability, when there are multiple machines, if preemption is not allowed, most problems are NP-hard. When preemption is allowed and the machines have limited availability constraints, the makespan and lateness problem are shown to be both solvable in P by Liu and Sanlaville [10]; additionally if the number of available machines does not go down by 2 within a period of $p_{\max}$, Leung and Pinedo [9] solved the total completion time minimization problem using preemptive SPT. An online version of makespan minimization has been considered by Albers and Schmidt [1].

New contributions. We note that if a multicriteria problem is NP-hard when the machines are constantly available, then it is also NP-hard when the machines have limited availability. Moreover, if a single criterion problem is NP-hard with limited machine availability, the multiple criteria problem with this single criterion as primary criterion is also NP-hard. However, for the problems which are polynomially solvable in the area of multicriteria scheduling or scheduling with limited machine availability, their complexity becomes open when we consider simultaneously multicriteria scheduling and limited machine availability constraint.

In this paper, we mainly study three such problems (1) $P_{1,1} \mid r-a, prmt \mid \sum C_j/C_{\max}$; (2) $P_{1,1} \mid r-a, prmt \mid C_{\max}/\sum C_j$; (3) $P_2 \mid r-a, prmt \mid C_{\max}/\sum C_j$ with a single zero-availability interval, $[t, t+x]$. Our first contribution is to show that the existing optimal algorithms in bicriteria scheduling or scheduling with machine availability constraint cannot be applied directly to our problems. Then we show all these problems are still in P by developing optimal algorithms. One intermediate result for problem (3) is an optimal algorithm for the problem $P_2 \mid r-a, prmt \mid \sum C_j$ with a single zero-availability interval.

All our algorithms are efficient, running in $O(n \log n)$ time. Besides they are quite simple and easy to implement. These features are very important in reality when we want to apply the algorithms in industry. On the other hand, the proofs of the optimality of our algorithms are very technical and quite involved.

Organization. Our paper is organized as follows. In Section 2, we show existing algorithms cannot be applied to solve our problems and give some preliminary results. In Section 3, we study $P_{1,1} \mid r-a, prmt \mid \sum C_j/C_{\max} \le T$, where T is a constant greater than or equal to $C^*_{\max}$. In Section 4, we study $P_{1,1} \mid r-a, prmt \mid C_{\max}/\sum C_j$. In Section 5, we first give an optimal algorithm for $P_2 \mid r-a, prmt \mid \sum C_j$ with a single zero-availability interval, and based on this result, we then give an optimal algorithm for $P_2 \mid r-a, prmt \mid C_{\max}/\sum C_j$ with a single zero-availability interval. In Section 6, we draw the conclusion.

2. Preliminaries

In this section, we first use a simple example to illustrate that the existing optimal algorithms cannot be applied to the first two problems we study. The example consists of three jobs, $p_1 = 10, p_2 = 5, p_3 = 3$; and two machines where machine 2 is unavailable during the interval [2, 4]. The optimal schedules for different criteria of these three jobs are shown in Fig. 1.

- With only one criterion $C_{\max}$: the problem is solved by Liu and Sanlaville [10] using LRPT (Largest Remaining Processing Time first) rule and processor sharing procedure (1995)). The optimal schedule is shown in Fig. 1(a).
- With only one criterion $\sum C_j$: the problem is solved by Leung and Pinedo [9] using SPT (Shortest Processing Time first) rule. The optimal schedule is shown in Fig. 1(b).
- With two criteria $\sum C_j/C_{\max}$, the optimal schedule is shown in Fig. 1(c). Apparently, neither 1(a) nor (b) is optimal, and we cannot apply the results of Leung and Pinedo [8] who solved this problem when machines are always available.
- With two criteria $C_{\max}/\sum C_j$, the optimal schedule is shown in Fig. 1(d). Again, neither 1(a) nor 1(b) is optimal, and we cannot apply the results of Leung and Young [7] who solved this problem when machines are always available.

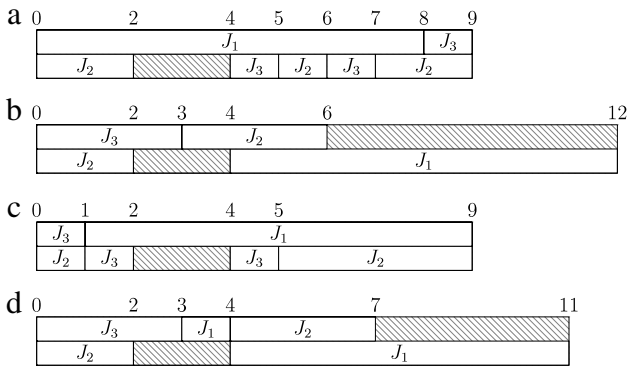

Fig. 1. Schedules of 3 jobs on 2 machines: machine 2 is unavailable during the interval $[2, 4]$; $p_1 = 10, p_2 = 5, p_3 = 3$. (a) Optimal schedule for makespan generated by LRPT rule and processor sharing; (b) Optimal schedule for total completion time generated by SPT rule; (c) Optimal schedule for total completion time subject to the constraint that makespan is minimum; (d) Optimal schedule for makespan subject to the constraint that total completion time is minimum.

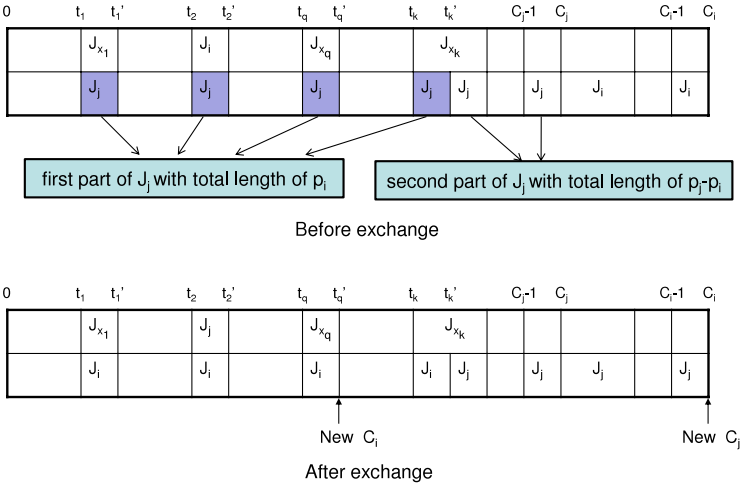

Fig. 2. Illustration of Lemma 1.

In the following, we will introduce PSPT rule and three lemmas which will facilitate us to solve our problems.

PSPT: preemptive SPT rule, i.e., at any time, if a machine becomes available, the job with the minimum remaining time get scheduled. Throughout this paper, we assume jobs $J_1, J_2, \ldots, J_n$ are indexed in nondecreasing order of their processing time.

Lemma 1. *For two machines and each of the objectives $\sum C_j$, $C_{\max}$, $\sum C_j/C_{\max}$ and $C_{\max}/\sum C_j$, there is a corresponding optimal schedule such that if $p_i < p_j$, then $C_i \leq C_j$.*

Proof. We prove by contradiction. Suppose in an optimal schedule for some objective above, there are two jobs J_i and J_j such that $p_i < p_j$ and $C_i > C_j$. We can divide all the intervals that J_j is scheduled into two parts such that the total length of the intervals in the first part is p_i and the total length of the intervals in the second part is $p_j - p_i$. We exchange the first part of J_j with J_i to form a new schedule (see Fig. 2 for an illustration). In this way, the completion time of job J_i is decreased by more than $C_i - C_j$, and the completion time of job J_j is increased by $(C_i - C_j)$ and all other jobs have same completion time as before. This is a contradiction to the assumption that the original schedule is optimal. □

Lemma 2. *For two machines with no zero-availability interval, PSPT generates an optimal schedule for $\sum C_j$.*

Proof. By the above lemma, we can assume that the jobs finish in SPT order in an optimal schedule. It is obvious that PSPT rule generates a schedule where the jobs finish in SPT order. To show that this schedule is optimal, we describe how to convert an optimal schedule into the schedule generated by PSPT rule without increasing the total completion time.

Suppose that in an optimal schedule, the jobs are not scheduled in PSPT order. Let t be the earliest time such that $p_i < p_j$, J_j is scheduled at t but J_i is not. We have two cases:

1. Job J_j is not scheduled at $[C_i - 1, C_i]$. For this case, we can get a new schedule by exchanging job J_i at $[C_i - 1, C_i]$ and job J_j at $[t, t + 1]$. In this way, C_i is decreased by at least 1, and C_j and all other jobs have same completion time as before, see Case 1 of Fig. 3;

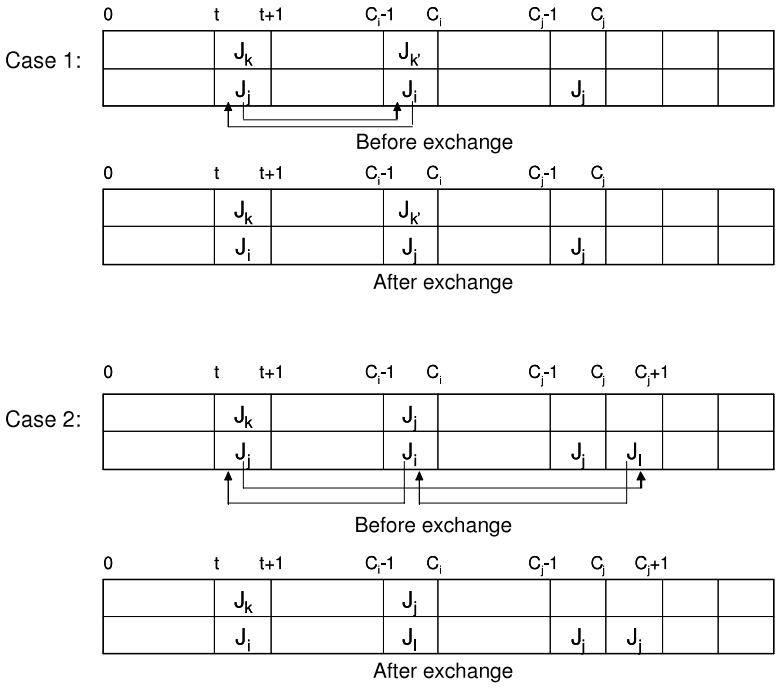

Fig. 3. Illustration of Lemma 2.

2. Job J_j is scheduled at $[C_i - 1, C_i]$. Let J_l be a job scheduled at $[C_j, C_j + 1]$; if there is no such job, we assume J_l is a dummy job scheduled at $[C_j, C_j + 1]$ (this is always possible when there is no zero-availability interval). We can get a new schedule without increasing the total completion time (but may increase the makespan) as follows: move job J_j at $[t, t + 1]$ to $[C_j, C_j + 1]$, move J_i at $[C_i - 1, C_i]$ to $[t, t + 1]$, and move J_l at $[C_j, C_j + 1]$ to $[C_i - 1, C_i]$. In this way, C_i is decreased by at least 1, C_j is increased by 1 and C_l is not increased, all other jobs have same completion time as before, see Case 2 of Fig. 3.

If we repeat the above conversion procedure, we get the schedule in PSPT rule at the end. □

Lemma 3. *Let A be an optimal algorithm for an objective from* $\sum C_j$, $C_{\max}$, $\sum C_j/C_{\max}$ *and* $C_{\max}/\sum C_j$ *on two machines with no zero-availability interval. Then A also generate an optimal schedule for the same objective on two machines with a single zero-availability interval* $[t, t + x]$, *if no job can finish before t in any schedule.*

Proof. For two machines with a single zero-availability interval $[t, t+x]$, if we remove the zero-availability interval, we get a new machine environment. Observe that there is one to one correspondence between schedules in the new environment and those in the original environment: let any S be a schedule in the original machine environment, if we simply remove the zero-availability interval, we get a new schedule $\hat{S}$ for the new environment. Furthermore, if no job can finish before t in any schedule, then we have, $C_{\max}(S) = C_{\max}(\hat{S}) + x$; and $\sum C_j(S) = \sum C_j(\hat{S}) + nx$. Thus the optimal schedule in the original environment must correspond to the optimal schedule in the new environment which can be generated by Algorithm A. □

3. $P_{1,1} \mid r - a, prmt \mid \sum C_j/C_{\max} \leq T$

In this section, we study the problem $P_{1,1} \mid r - a, prmt \mid \sum C_j/C_{\max} \leq T$, where T is a constant greater than or equal to the optimal makespan $C^*_{\max}$. We give an optimal polynomial time algorithm for this problem. The basic idea of our optimal algorithms is to schedule the jobs one by one using PSPT rule, subject to the condition that job J_n can finish by T.

Algorithm 1. Let S be an empty schedule.
Set $overlap := false$ and $i := 1$
While ($i \leq n$ and $overlap = false$)
Add job J_i to S by PSPT rule.
If J_n can be fully scheduled backwards from T using PSPT rule in S
$i := i + 1$
Else
$overlap := true$

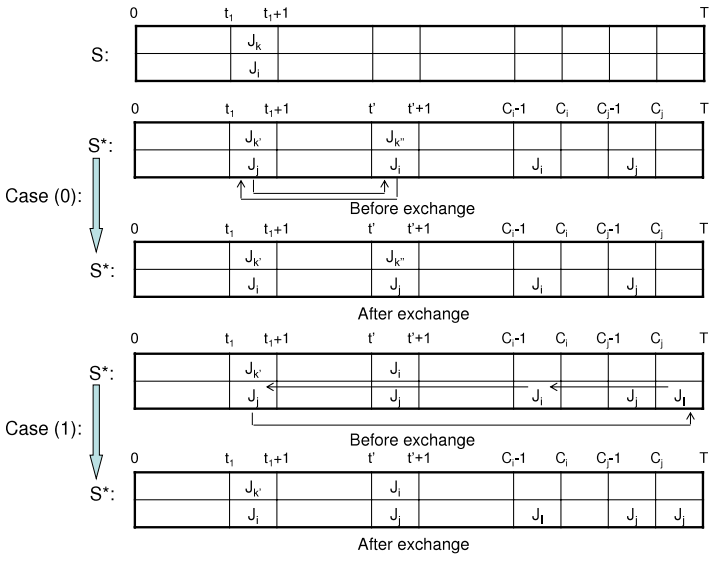

Fig. 4. Illustration Case (0) and Case (1) of Theorem 4.

Schedule as much of J_n as possible backwards from T using PSPT rule in S.
Let l be the length of the unscheduled part of J_n.
Backwards from the time that J_i finishes, find intervals of total length l where J_i is scheduled but J_n is not.
Schedule the remaining of J_n in these intervals.
Reschedule the part of J_i in these intervals as early as possible.

If $i < n$, add jobs $J_{i+1}, \ldots, J_{n-1}$ in this order to S after J_i.
Return S

Theorem 4. *Algorithm 1 find the optimal schedule for* $P_{1,1} \mid r - a, prmt \mid \sum C_j/C_{\max} \leq T$.

Proof. If *overlap* is never reset to be true in the algorithm, then it is easy to see that all the jobs are scheduled before T using PSPT rule. By Lemma 2, the final schedule S must be optimal.

Otherwise *overlap* is set to be true when some i, $i < n$, is processed. Note that after J_n is completely scheduled and J_i is rescheduled, at any time after J_i finishes and before T, at most one machine is available, and the total length of available intervals before T must be greater than or equal to the total processing time of $J_{i+1}, \ldots, J_{n-1}$. Thus the schedule S is feasible.

Now we prove that S is optimal by showing that there exists an optimal schedule that schedules the jobs same as S; i.e., smaller jobs are scheduled as early as possible subject to the condition that job J_n can finish by time T. Suppose that there is one optimal schedule S^*, that does not schedule the smaller jobs as early as possible. Then there exist a time t_1 and two jobs J_i and J_j such that (1) $i < j$; (2) i is the smallest index such that at t_1, J_i is scheduled in S but not in S^*; (3) j is the largest index such that at t_1, J_j is scheduled in S^* but not in S.

Let t_1 be the earliest such time. Then the schedule before t_1 in S must be the same as in S^* and all the jobs $J_1, J_2, \ldots,$ and J_{i-2} must have been completed. By Lemma 1, we have $C_i \leq C_j$ in S^*.

Case 0: In S^*, there is a time instant $t' > t_1$ such that J_i is scheduled but J_j is not scheduled, then we can exchange J_j at t_1 with J_i at t', see Case (0) of Fig. 4. The completion time of J_i and J_j are not increased and the completion time of other jobs are unchanged, so the total completion time is not increased.

Next we assume t' does not exist in S^*; i.e. for S^*, at any time after t_1, if J_i is scheduled, then J_j must also be scheduled, which we denote as property 1. We have the following two cases.

Case 1: $C_j \neq T$. Since there is a machine always available, then there must exist a job J_l ($l \neq i$) scheduled at $[C_j, C_j + 1]$. We can convert S^* as follows: move J_j at $[t_1, t_1 + 1]$ to $[C_j, C_j + 1]$, move J_i at $[C_i - 1, C_i]$ to $[t_1, t_1 + 1]$, and move J_l at $[C_j, C_j + 1]$ to $[C_i - 1, C_i]$. In this way, C_i is decreased by at least 1, C_j is increased by 1 and C_l is not increased, all other jobs have same completion time as before, and the total completion time is not increased. See Case (1) of Fig. 4.

Case 2: $C_j = T$. By Lemma 1, either $j = n - 1$ or $j = n$.

Case (2a): $j = n - 1$.

We prove this case does not exist by contradiction. By Property 1, we know at any time after t_1, if J_i is scheduled, J_{n-1} must also be scheduled. Since $p_{n-1} \leq p_n$, there must exist a time instant t_2, $t_1 < t_2 < T$, such that J_n is scheduled at t_2, and J_{n-1} and J_i are not scheduled at t_2. See Case (2a) of Fig. 5. We can convert S^* as follows: move J_{n-1} at $[t_1, t_1 + 1]$ to $[t_2, t_2 + 1]$, move J_i at $[C_i - 1, C_i]$ to $[t_1, t_1 + 1]$, and move J_n at $[t_2, t_2 + 1]$ to $[C_i - 1, C_i]$. In this way, C_i is decreased by at least 1, all other jobs have same completion time as before, and the total completion time is decreased. This contradicts to the fact that S^* is optimal. So j could not be $n - 1$.

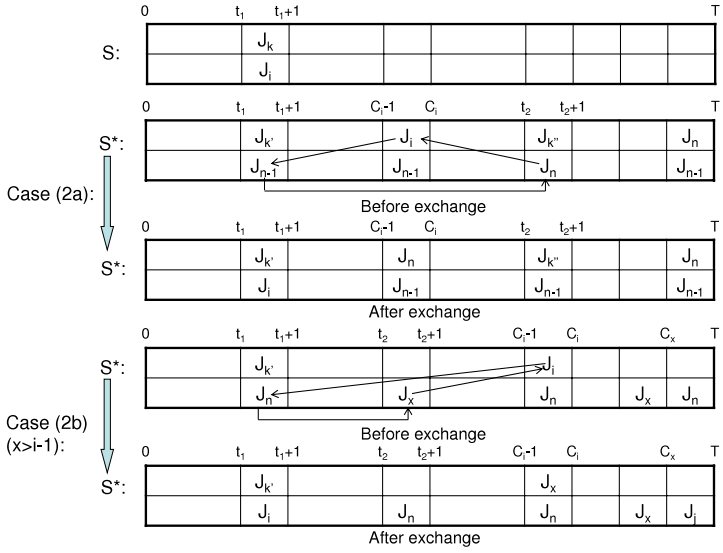

Fig. 5. Illustration of Case 2 of Theorem 4.

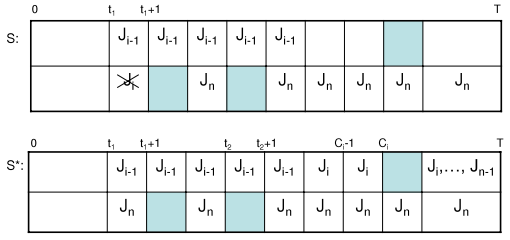

Fig. 6. Illustration of Case 2b ($x = i - 1$) of Theorem 4.

Case (2b): $j = n$.

We prove by contradiction that this case does not exist either. Since J_n is not scheduled at t_1 in S, there must exist a time instant $t_2 > t_1$ such that J_n is not scheduled at t_2 in S^*. Because of property 1, J_i is not scheduled at t_2 either. Let J_x be the job that is scheduled at t_2. Then $x \geq i-1$ and $x \neq i$. Suppose $x > (i-1)$ which implies $C_x > C_i$, see Case (2b) of Fig. 5. We can convert S^* as follows: move J_n at $[t_1, t_1 + 1]$ to $[t_2, t_2 + 1]$, move J_i at $[C_i - 1, C_i]$ to $[t_1, t_1 + 1]$, and move J_x at $[t_2, t_2 + 1]$ to $[C_i - 1, C_i]$. In this way, C_i is decreased by at least 1, all other jobs have same completion time as before, and the total completion time is decreased, which is in contradiction to the fact that S^* is optimal. Thus $x = i-1$. This means that at any time t_2 when J_n is not scheduled in S^*, only job J_{i-1} can be scheduled at the time. This implies during the interval $[t_1 + 1, C_{i-1}]$ in S^*, either J_n is scheduled, or only one machine is available and J_{i-1} is scheduled; and after C_{i-1}, J_n is continuously scheduled. Furthermore, we can assume that J_{i-1} is scheduled before J_i, which is before $J_{i+1}, \ldots, J_{n-1}$, see Fig. 6. Otherwise, we can do simple exchange without increasing the total completion time. However, in this case, if we schedule J_i at t_1, it is impossible to finish J_n by T, this contradicts to the fact that J_i is indeed scheduled at t_1 in S.

To summarize, if there is an optimal schedule S^* does not schedule the jobs as early as possible, then we can find a time t_1 such that J_i is scheduled in S but not in S^* and $J_j, j > i$, is scheduled in S^* but not in S. Then we know either Case 0 or Case 1 must happen, and we can convert S^* so J_i is also scheduled at t_1, without increasing the total completion time. We can repeat this process until S^* is completely the same as S. This means that S is also optimal. □

4. $P_{1,1} \mid r - a, prmt \mid C_{\max} / \sum C_j$

In this section, we show that the problem $P_{1,1} \mid r - a, prmt \mid C_{\max} / \sum C_j$ is in P by developing a polynomial time optimal algorithm. The basic idea of our optimal algorithms is first scheduling the jobs one by one using PSPT rule, which guarantees the optimal total completion time; then adjusting the jobs J_{n-1} and J_n to further minimize the makespan.

Algorithm 2. Let S be the PSPT schedule of $J_1, \ldots, J_n$

Let C_{n-1} and C_n be the completion time of the jobs J_{n-1} and J_n in S, respectively.

Let $[C_{n-1}, C_{n-1} + \delta_1]$ be the **continuous** idle interval after C_{n-1} and before C_n, i.e. for any time t, $C_{n-1} \leq t < C_{n-1} + \delta_1$, there is one machine available during $[t, t + 1]$.

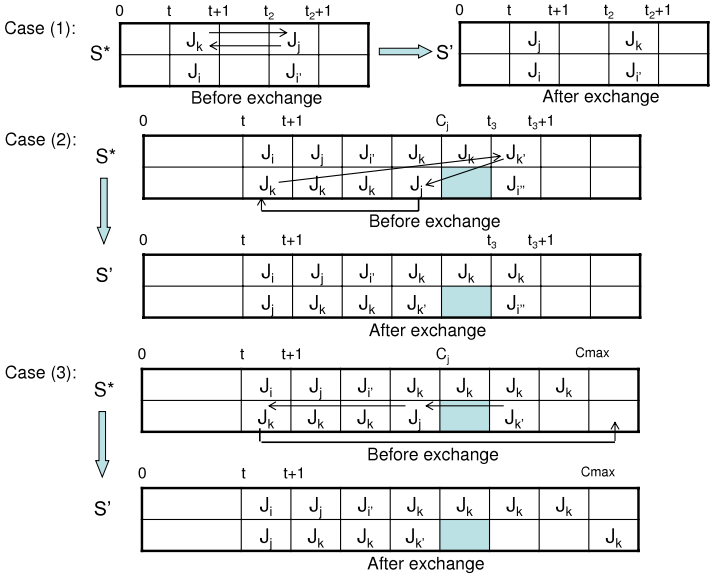

Fig. 7. Illustration of Theorem 5.

Let δ_2 be the total length of the intervals such that job J_{n-1} is scheduled during the interval but J_n is not.
Let $\delta = \min(\delta_1, \delta_2, \frac{1}{2}(C_n - C_{n-1}))$.
Reschedule job J_{n-1} of length δ from those intervals during which job J_n is not scheduled, to $[C_{n-1}, C_{n-1} + \delta]$, reschedule the part of J_n from $[C_n - \delta, C_n]$ to those intervals where J_{n-1} is rescheduled.
Return S

Theorem 5. *Algorithm 2 is optimal for* $P_{1,1} \mid r - a, prmt \mid C_{\max}/\sum C_j$.

Proof. We first show that the schedule S is feasible: i.e, the total completion time is equal to C^*. This is so initially since PSPT generates the optimal schedule for $\sum C_j$. We then minimize the makespan by preempting $n-1$, so that the completion time of J_{n-1} is increased by δ, and the completion time of J_n is decreased by δ, thus the total completion time is unchanged.

Now we show that there is an optimal schedule such that three properties hold: (1) the jobs finish in SPT order; (2) the jobs $J_1, J_2, \ldots, J_{n-2}$ are scheduled in PSPT rule; (3) for any job J_j, $1 \le j \le n-2$, J_j will not be preempted by J_{n-1} or J_n such that at some time t J_{n-1} and/or J_n are scheduled, while J_j is not finished but not scheduled at t.

By Lemma 1, property (1) is obviously true. By the same argument in the proof of Lemma 2, property (2) is true. Now we consider property (3). Suppose that an optimal schedule S^* has properties (1) and (2) but not property (3). Then there is a time t in S such that $J_j, j \le n-2$, is not scheduled at t, but job J_k, $k = n-1, n$, is scheduled. Let t be the earliest such time. We show that we can convert S^* into a new schedule S' such that J_j is scheduled at time t in S' without increasing the total completion time and the makespan. There are three cases (see Fig. 7):

Case 1: There is a time $t_2 > t$ such that J_j is scheduled but J_k is not. We exchange J_k at $[t, t+1]$ with J_j at $[t_2, t_2+1]$. This will not increase the completion time of J_j and J_k.

Case 2: Time t_2 does not exist, i.e, whenever J_j is scheduled after t, J_k is also scheduled in S^*. Find the earliest time t_3, $C_j \le t_3 \le C_{\max}$, such that J_k is not scheduled at t_3, and a job $J_{k'}$, $k' \ne j$, is scheduled at t_3. If t_3 exist, J_k must be continuously scheduled between C_j and t_3 in S, thus $C_k \ge t_3$ in S^*. We do triple exchange: move $J_{k'}$ from $[t_3, t_3+1]$ to $[C_j - 1, C_j]$, move J_j from $[C_j - 1, C_j]$ to $[t, t+1]$, move J_k from $[t, t+1]$ to $[t_3, t_3+1]$. In this way, the completion time of J_j is decreased by at least 1, the completion time of J_k is increased by at most 1, and the completion time of $J_{k'}$ is not increased. Thus neither the total completion time, nor the makespan is increased.

Case 3: Neither t_2, nor t_3 exists, which means that whenever J_j is scheduled after t, J_k is also scheduled, and J_k is continuously scheduled from time C_j to $C_{\max}$, so $C_k = C_{\max}$. We show this case is impossible.

Since $j \le n-2$, by Lemma 1, besides job J_k, there is at least one other job $J_{k'}$ that has completion time greater than or equal to C_j. Furthermore, $J_{k'}$ cannot be scheduled at $[C_j - 1, C_j]$ since J_k and J_j are scheduled at this time. Thus $C'_k > C_j$. Now we convert S as follows: move the last time unit of $J_{k'}$ to $[C_j - 1, C_j]$, move the last time unit of J_j from $[C_j - 1, C_j]$ to $[t, t+1]$, move J_k from $[t, t+1]$ to $[C_{\max}, C_{\max}+1]$. The completion time of J_j and $J_{k'}$ are both decreased by at least 1, and the completion time of J_k is increased by exactly 1. So we get a schedule with a smaller total completion time, which contradicts to the fact that S is an optimal schedule.

By repeatedly applying the above conversion procedure, finally we get a schedule so that properties (1)–(3) are all satisfied. Properties (2) and (3) implies there is an optimal schedule such that jobs $J_1, J_2, \ldots, J_{n-2}$ are scheduled in PSPT rule, and job J_n may preempt only job J_{n-1} to minimize makespan.

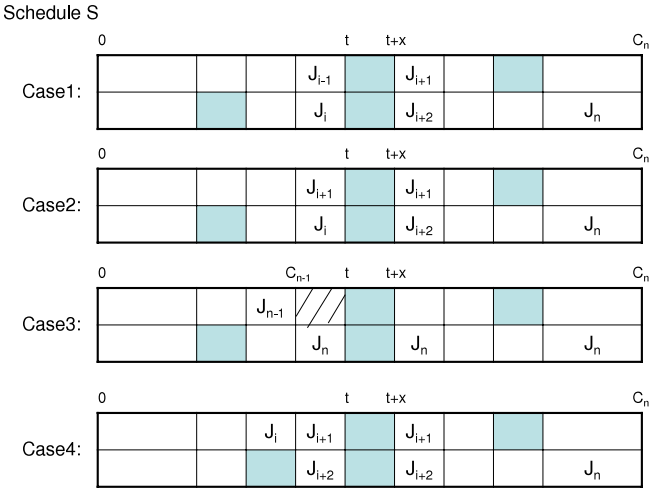

Fig. 8. Illustration of four cases for Algorithm 3.

Now we are ready to show that Algorithm 2 returns an optimal schedule. Since the first $n-2$ jobs are scheduled in PSPT rule, all we need to show is that $\delta = \min(\delta_1, \delta_2, \frac{1}{2}(C_n - C_{n-1}))$ is the upper bound that J_{n-1} can be preempted by J_n in order to improve the makespan and keep the total completion time optimal. δ_1 is an upper bound of preemption since to make the total completion time minimum, the increased amount of completion time of J_{n-1} has to be equal to the decreased amount of completion time of J_n (note that job J_n is scheduled continuously after C_{n-1}); δ_2 is an upper bound since job J_n cannot overlap itself; $\frac{1}{2}(C_n - C_{n-1})$ is an upper bound for preemption since it is a lower bound of the makespan. □

5. $P_2 \mid r - a, prmt \mid C_{\max}/\sum C_j$

In this section, we study the problem $P_2 \mid r - a, prmt \mid C_{\max}/\sum C_j$. We assume that there is only one zero-availability interval, $[t, t+x]$, and at any other time, there is at least one machine available. Before we give the algorithm for this problem, we first give the optimal algorithm for $P_2 \mid r - a, prmt \mid \sum C_j$, which will be used to solve $P_2 \mid r - a, prmt \mid C_{\max}/\sum C_j$.

5.1. $P_2 \mid r - a, prmt \mid \sum C_j$

Algorithm 3.
1. Let S denote the PSPT schedule of jobs $J_1, J_2, \ldots, J_n$. Let $J_1, J_2, \ldots, J_i$ be the jobs finish before t.
2. Case 1: No job spans the zero-availability interval, $[t, t+x]$ (see Fig. 8 Case 1), return S.
3. Case 2: Single job, J_{i+1} ($i+1 \neq n$), spans $[t, t+x]$ (see Fig. 8 Case 2), return S.
4. Case 3: Single job, J_n, spans $[t, t+x]$ (see Fig. 8 Case 3):
   (a) Let $[C_{n-1}, C_{n-1} + \delta_1]$ be the total length of idle intervals between C_{n-1} and t. Let $\delta_2 = C_n - (t+x)$.
   (b) If $\delta_1 < \delta_2$, then return S.
   (c) Otherwise
      i. Let S' be the schedule obtained by applying Algorithm 1 to all jobs with $T = t$.
      ii. Return the schedule S or S' whichever has a smaller total completion time.
5. Case 4: Two jobs, J_{i+1}, J_{i+2}, span the interval $[t, t+x]$ (see Fig. 8 Case 4).
   (a) Let δ_1 be the length of the part of J_{i+2} that is scheduled before t, which may span several intervals. Let δ_2 be the length of the part of J_{i+1} that is scheduled after $(t+x)$ in S.
   (b) If $\delta_1 < \delta_2$, then return S.
   (c) Otherwise
      i. Apply Algorithm 1 to jobs $J_1, J_2, \ldots, J_{i+1}$ with $T = t$.
      ii. Schedule the jobs $J_{i+2}, \ldots, J_n$, using PSPT rule.
      iii. Let S' be the schedule obtained from above.
      iv. Return S or S' whichever has a smaller total completion time.

Theorem 6. *Algorithm 3 is optimal for $P_2 \mid r - a, prmt \mid \sum C_j$ when there is a single zero-availability interval.*

Proof. For schedule S, we use $\hat{S}$ to denote its corresponding schedule without the zero-availability interval $[t, t+x]$. Since only the first i jobs finish before t, the total completion time is $C(S) = C(\hat{S}) + (n-i) \cdot x$. Also by Lemma 2, $\hat{S}$ is optimal if the zero-availability interval $[t, t+x]$ is removed.

Now we prove that Algorithm 3 returns the optimal schedule in all cases.

Case 1: No job spans the zero-availability interval. This means the first i jobs fit exactly the available intervals before t. Since the first i jobs are the shortest jobs, for any optimal schedules S^*, there are at least $(n-i)$ jobs scheduled after zero-availability interval. Let $\hat{S^*}$ be the schedule that corresponds to S^* without the interval $[t, t+x]$. The total completion time is $C(S^*) \geq C(\hat{S^*}) + (n-i)\cdot x$. By Lemma 2, $C(\hat{S}) \leq C(\hat{S^*})$, thus $C(S) \leq C(S^*)$ and S is an optimal schedule.

Case 2: Single job, single job J_{i+1}, $i+1 \neq n$, spans $[t, t+x]$. As Case 1, any optimal schedule has at least $(n-i)$ jobs finish after t, thus we can show S is optimal.

Case 3: Single job, J_n, spans $[t, t+x]$. The total completion time of S is thus $C(\hat{S}) + x$.

We have two subcases. In the first subcase, $\delta_1 < \delta_2$, this means that any optimal schedule S^* has to have at least one job finish after $t+x$, thus has a total completion time at least $C(\hat{S^*}) + x \geq C(\hat{S}) + x = C(S)$. So S is optimal. In the second subcase, $\delta_1 \geq \delta_2$, one may have a better schedule S' in which job J_n preempts some jobs and finishes before t. By Theorem 4, Algorithm 1 returns a schedule that minimizes the total completion time subject to all jobs finish before t. On the other hand, S minimizes the total completion time subject to at least one job finishes after t. So the optimal schedule of our problem must be either S or S' depending on which has the smaller total completion time.

Case 4: Two jobs, J_{i+1}, J_{i+2}, span $[t, t+x]$.

Depending on whether the first $i+1$ jobs can be scheduled so that they finish before t, we also have two subcases. If $\delta_1 < \delta_2$, any schedule can have at most i jobs finish before t. Similar to Case 3, one can show S is optimal.

Otherwise, some schedules may have $(i+1)$ jobs finish before t. We claim that among such schedules, S' has the minimum total completion time. For convenience, given any partial schedule, we use "profile" to describe the available intervals of the machines. We are specifically interested in the profile, after the first $i+1$ jobs are scheduled. In S', after the first $i+1$ jobs are scheduled, the available intervals before t are all on a single machine and have a total length of $\delta_1 - \delta_2$.

For any schedule S^* such that $i+1$ jobs are finished before t, by Lemma 1, these jobs must be the first $(i+1)$ jobs. After these jobs are scheduled, the available intervals after t are exactly the same as that of S'; the available intervals before t have a total length same as that of S', $\delta_1 - \delta_2$, and these available intervals may be all on a single machine, or may be on two machines. No matter what profile S^* has after the first $i+1$ jobs are scheduled, corresponding to the schedule of the remaining jobs $J_{i+2}, \ldots, J_n$, we can get a new schedule $\hat{S^*}$ of the remaining jobs that fits into the profile of S' without changing the completion time of any job in S^*: Let $J_{i_1}, \ldots, J_{i_u}$ be the partial jobs which are scheduled before t in S^*. Given the profile of S', we simply schedule the same amount of $J_{i_1}, \ldots, J_{i_u}$ to the available intervals before t at any order; and schedule the remaining jobs exactly the same way as S^*. So we must have $\sum C_j(\hat{S^*}) = \sum C_j(S^*)$ and $\hat{S^*}$ is a feasible schedule for the profile of S' after the first $i+1$ jobs are scheduled.

On the other hand, given the profile of S' after the first $i+1$ jobs are scheduled, by Lemma 3, S' optimally scheduled the remaining jobs for the objective of total completion time. By Lemma 2, the first $(i+1)$ jobs are also scheduled in S' so they have the minimum total completion time. Therefore S' must be optimal. □

5.2. $P_2 \mid r-a, prmt \mid C_{\max}/\sum C_j$

Based on the optimal algorithm for $P_2 \mid r-a, prmt \mid \sum C_j$, we give our optimal algorithm for $P_2 \mid r-a, prmt \mid C_{\max}/\sum C_j$ as follows assuming there is a zero-availability interval, $[t, t+x]$.

Algorithm 4. 1. Apply Algorithm 3, and get a schedule S.
2. If J_{n-1} completes before t and J_n completes after $(t+x)$, let $[C_{n-1}, C_{n-1}+\delta_1]$ be the continuous idle interval before t, let δ_2 be the total length of intervals where $(n-1)$ is scheduled but n is not, let $\delta = \min(\delta_1, \delta_2, \frac{1}{2}(C_n - C_{n-1} - x))$.
3. Else, J_{n-1} and J_n both complete before or both after $(t+x)$.
   Let $[C_{n-1}, C_{n-1}+\delta_1]$ be the continuous idle interval after C_{n-1} and before C_n, let δ_2 be the total length of intervals where J_{n-1} is scheduled but J_n is not, let $\delta = \min(\delta_2, \delta_1, \frac{1}{2}(C_n - C_{n-1}))$.
4. Reschedule the part of J_{n-1} of length δ from those intervals during which job J_n is not scheduled, to $[C_{n-1}, C_{n-1}+\delta]$, reschedule the part of J_n from $[C_n - \delta, C_n]$ to those intervals where J_{n-1} was rescheduled. Let the new schedule be S'.
5. Return S'.

Theorem 7. *Algorithm 4 is optimal for $P_2 \mid r-a, prmt \mid C_{\max}/\sum C_j$ when there is a single zero-availability interval.*

Proof. First of all, the schedule S produced by Algorithm 3 is optimal for the total completion time and the reschedule of J_n and J_{n-1} does not change the total completion time, thus S' is feasible.

Next we prove that S' is optimal. We have to look at how S is generated in Algorithm 3. There are two cases.

Case 1: S is the schedule produced by PSPT rule.

In this case, Algorithm 4 is the same as Algorithm 2. If $C_{\max}(S) \leq t$, by Theorem 5, S^* must be optimal. So we assume $C_{\max}(S) \geq t+x$, i.e. at least one job that finishes after the zero-availability interval. From the proof of

Theorem 6, we know to minimize the total completion time, every schedule must have at least one job finishes after the zero-availability in this case. Let S^* be the optimal schedule. Then we must have $C_{\max}(S^*) \geq t + x$.

If we remove the interval $[t, t+x]$ from S^*, we get a schedule whose makespan is $C_{\max}(S^*) - x$. If we remove zero-availability interval from S', then we get an optimal schedule for the new machine environment whose makespan is $C_{\max}(S') - x$. So we must have $C_{\max}(S^*) - x \geq C_{\max}(S') - x$, and therefore, $C_{\max}(S^*) \geq C_{\max}(S')$, which means that S' is optimal.

Case 2: S is not the schedule produced by PSPT rule, then S must be produced from either Case 3 or Case 4 of Algorithm 3.

- S is generated from Case 3 of Algorithm 3, which calls Algorithm 1.

  Then Algorithm 4 reschedules J_{n-1} and J_n in S to get S' which has the same total completion time as S. We prove S' is optimal by showing that, any schedule whose makespan is less than $C_{\max}(S')$, must have a total completion time greater than that of S' and S.

  Let J_i be the job which is being scheduled when overlap is set to be true in Algorithm 1, and let l be the length of the unscheduled part of J_n.

  Let $T = C_{\max}(S') - \epsilon$ and apply Algorithm 1. Let S'' be the produced schedule and we compare its total completion time with that of S. When S'' is produced by Algorithm 1, we must have: either (a) when overlap is set to be true, J_i is being considered and the length of the unscheduled part of J_n is $l + \epsilon$; or (b) when overlap is set to be true, job $J_j, j < i$, is considered. For (a), the completion times of jobs $J_i, \ldots, J_{n-1}$ are all increased by at least ϵ compared with that of S, and the completion time of job J_n is decreased by ϵ. Thus the total completion time is more than that of S. For (b), compared with that of S, the completion time of J_n is decreased by ϵ; in S J_n is continuously scheduled after J_i is finished while in S'', J_n is continuously scheduled after J_j is finished. Because J_n finishes ϵ time units earlier and we have ϵ more units of J_n scheduled before J_i, the completion time of J_i is increased by at least ϵ; and the completion time of J_j is increased since we have to reschedule J_j. Overall the total completion time is more than that of S.
- S is generated from Case 4 of Algorithm 3, which calls Algorithm 1.

  From the proof of Theorem 6, in order to minimize the total completion time, the optimal schedule must have the first $i+1$ jobs finish before t. As in the proof of Theorem 6, no matter how the jobs $J_{i+2}, \ldots J_n$, are scheduled, we can get a corresponding schedule that fits into the profile of S' after the first $i + 1$ jobs finishes without increasing the total completion time and the makespan. Given the profile with the first $i + 1$ jobs scheduled, our problem becomes minimizing makespan of the remaining jobs subject to the total completion time of the remaining jobs is minimum. Since no remaining job can finish before the zero-availability interval, by Lemma 3, we can use Algorithm 2 to solve the subproblem, this is exactly what Algorithm 3 does. Thus S' is optimal. □

6. Conclusion

In this paper we study the problems of two parallel machine bicriteria scheduling subject to limited machine availability concerned with makespan and total completion time. When there is no zero-availability interval, we give the optimal polynomial time algorithms for the objective of makespan subject to total completion time is minimized and the objective of total completion time subject to makespan is minimized.

When both machines are not available at time $[t, t + x]$, we give an optimal algorithm for the objective of makespan subject to total completion time is minimized. And one intermediate result for this problem is the optimal algorithm for the problem $P_2 \mid r - a, prmt \mid \sum C_j$ when both machines are not available at time $[t, t + x]$.

When there are two or more zero-availability intervals, one can easily show that the optimal algorithm for the problem $P_2 \mid r - a, prmt \mid \sum C_j$ with single zero-availability does not work any more and seems difficult to be extended. So this still leaves the complexity of the general problem, $P_2 \mid r - a, prmt \mid \sum C_j$, open. Consequently, the complexity status of $P_2 \mid r - a, prmt \mid C_{\max}/\sum C_j$ is also open when there are multiple zero-availability intervals. Another open problem is the complexity of $P_2 \mid r - a, prmt \mid \sum C_j/C_{\max}$ with one or more zero-availability intervals. For the multiple parallel machines' environment, it will be interesting to determine the complexity result of $P \mid r - a, prmt \mid C_{\max}/\sum C_j$ and $P \mid r - a, prmt \mid \sum C_j/C_{\max}$ when the number of available machines does not go down by 2 within a period of $p_{\max}$.

References

[1] S. Albers, G. Schmidt, Scheduling with Unexpected Machine Breakdowns, in: Randomization, Approximation, and Combinatorial Optimization (Algorithms and Techniques), vol. 1671, Springer, Berlin, Heidelberg, 2004, pp. 269–280.
[2] C.L. Chen, R.L. Bulfin, Complexity of single machine, multicriteria scheduling problems, European Journal of Operational Research 70 (1993) 115–125.
[3] P. Dileepan, T. Sen, Bicriteria static scheduling research for a single machine, OMEGA 16 (1988) 53–59.
[4] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling, a survey, Annals of Discrete Mathematics 5 (1979) 287–326.
[5] J.N.D. Gupta, J.C. Ho, S. Webster, Bicriteria optimisation of the makespan and mean flowtime on two identical parallel machines, Journal of Operational Research Society 51 (11) (2000) 1330–1339.
[6] J.A. Hoogeveen, Multicriteria scheduling, European Journal of Operational Research 167 (3) (2005) 592–623.
[7] J.Y-T. Leung, G.H. Young, Minimizing schedule length subject to minimum flow time, SIAM Journal on Computing 18 (2) (1989) 314–326.
[8] J.Y-T. Leung, M.L. Pinedo, Minimizing total completion time on parallel machines with deadline constraints, SIAM Journal on Computing 32 (2003) 1370–1388.

[9] J.Y-T. Leung, M.L. Pinedo, A Note on the scheduling of parallel machines subject to breakdown and repair, Naval Research Logistics 51 (2004) 60–72.
[10] Z. Liu, E. Sanlaville, Preemptive scheduling with variable profile, precedence constraints and due dates, Discrete Applied Mathematics 58 (1995) 253–280.
[11] H. Saidy, M. Taghvi-Fard, Study of scheduling problems with machine availability constraint, Journal of Industrial and Systems Engineering 1 (4) (2008) 360–383.
[12] G. Schmidt, Scheduling with limited machine availability, European Journal of Operational Research 121 (1) (2000) 1–15.
[13] V. T'kindt, J.C. Billaut, Multicriteria Scheduling: Theory, Models and Algorithms, Springer Verlag, Heidelberg, 2002.